

X.commerce Workshop – Java

Setup and Configuration

This lab assumes an intermediate level of experience with Java as a programming language and uses the Eclipse IDE throughout the examples. However, you should be able to complete the exercises with any Java development environment. You'll just need to be able to translate the Eclipse-specific instructions to your IDE of choice.

Also, any shell commands assume that you're working in a flavor of Unix (including Cygwin), but they should be easily translated into DOS commands, if you're working in Windows.

Beyond a Java 6 JDK and Eclipse (<http://www.eclipse.org/downloads>), all of the software that you need to complete this lab is provided in the X.commerce Developer Package, which you should have already installed, and the supplementary xdevcamp-java.zip file. Before we get started, please unzip the contents of the xdevcamp-java.zip file into your directory of choice, e.g. C:\xdevcamp.

Lab 1: Setup the Fabric

Setup Mongo

1. Create Mongo data directory: `mkdir /data/db`
2. Start MongoDB: `~/dev/mongodb-osx-x86_64-1.8.2/bin/mongod &`

Start the Fabric

1. Using Ant: From the root of the developer package, run: `ant xfabric.start`
2. Without Ant: `java -jar main-xfabric-0.5.4.jar`

Open the Manager UI

1. In your favorite modern web browser, open: <http://localhost:8079/>
2. Login with default credentials: `admin / password0`

Lab 2: Run AuctionHouse application

Create Auction Topics

1. Open the Manager UI
2. Click the "Topics" tab
3. Scroll down to the "Define Topic" form
4. Create the following topics:

Topic Name	Tenant Require?
/auction/ended	No
/auction/started	No
/auction/updated	No
/bid/accepted	Yes
/bid/extended	Yes
/bid/outbid	Yes
/bid/placed	Yes
/bid/rejected	Yes
/bid/won	Yes

Define Capability in the Manager

1. In the Capabilities drop-down, select "Sample Capability 1"
2. Enter **http://localhost:8888/** in the "Endpoint" field and then click "Update".

Subscribe Capability to Topics

1. Click on the "Subscriptions" tab
2. Enter **/bid/placed** in the "Topic Pattern" field and click "Subscribe"

Register Tenants with the Capability

1. Click on the "Authorization Info" tab
2. Ensure that "merchant 2" and "merchant 3" are in the "Authorizations" table.
3. If not, add them with the "Authorize Tenant" form.

Start AuctionHouse application

1. On the command-line, navigate to the folder where you unzipped the xdevcamp-auction-java.zip file.
2. Run

```
java -jar common/auctionhouse-0.1-bundle.jar
http://localhost:8080/ 8888
```

See Messages

1. Back in the Manager UI, click on the "Message Tracing" tab
2. In the "Find Recent Messages" form, click "Search"
3. You will see a list of message IDs, one for each message that the AuctionHouse app has published.

Lab 3: Create new capability

Create a new project folder

1. In a folder of your choosing, aka `<root-dir>`, run:
`mkdir <root-dir>/bidder`
2. Run: `mkdir <root-dir>/bidder/lib`
3. Copy files from `<assets-dir>/java/lib` to `<root-dir>/bidder/lib`

Create new project

1. In Eclipse, select the `File > New... > Other...` method item.
2. Choose "Java Project", then click "Next"
3. Set the "Name" field to `bidder`
4. Set the "Location" field to `<root-dir>/bidder`
5. Click "Finish"

Setup source folders


1. Right-click "bidder" project and select the `New > Source Folder` option
2. Enter `src` in the "Folder name" field and click "Finish"
3. Repeat with `avro-src` as the "Folder name"

Generate Avro source files

1. From the `<assets-dir>/java` directory, run:
 - `java -jar avro-tools-1.5.2.jar compile protocol ../common/Marketplace.avpr <root-dir>/bidder/avro-src/`
 - `java -jar avro-tools-1.5.2.jar compile protocol ../common/Auction.avpr <root-dir>/bidder/avro-src/`
2. Refresh the 'bidder' project in Eclipse to see the generated code.

Lab 4: Register capability with the Fabric

Register new capability with the Fabric

1. Open the Manager UI: <http://localhost:8079/> 
2. In the Capabilities drop-down, select the "Sample Capability 2" capability
3. Set the Endpoint field to `http://localhost:9000/` and then click "Update"

Subscribe Capability to Topics

1. Click on the "Subscriptions" tab
2. Enter the following topics in the "Topic Pattern" field and click "Subscribe":

○ <code>/auction/started</code>	○ <code>/bid/extended</code>
○ <code>/auction/updated</code>	○ <code>/bid/outbid</code>
○ <code>/auction/ended</code>	○ <code>/bid/rejected</code>
○ <code>/bid/accepted</code>	○ <code>/bid/won</code>

Register Tenants

1. Click on the "Authorizations" tab
2. Ensure that "merchant 2" and "merchant 3" are in the "Authorizations" table.
3. If not, add them with the "Authorize Tenant" form.

Validate that messages are being delivered

1. Click on the "Message Trace" tab
2. In the "Find Recent Messages" form, select capability "Sample Capability 2" and click "Search"
3. Click on one of the resulting messages to see details .
Note: Getting a 404 message status is expected because the server you are sending a message to doesn't know how to process it yet.

Lab 5: Start Receiving Messages

Note: Throughout the next two labs, you can use the key-combination Ctrl-Shift-O or Command-Shift-O to automatically resolve missing imports.

Import some helper classes

Copy the <assets-dir>/java/source/com and <assets-dir>/java/source/org folders into the <root-dir>/bidder/src folder.

```
org.apache.avro.file.SeekableByteArrayInput
com.x.devcamp.common.AvroEncDecoder
com.x.devcamp.common.MessagePublisher
com.x.devcamp.common.MessagePublishException
com.x.devcamp.common.XFabricPublisher
```

Create com.x.devcamp.bidder.AuctionConsole class

1. In Eclipse, right-click on the "bidder/src" folder and select the New > Class... option
2. Enter **com.x.devcamp.bidder** in the "Package" field
3. Enter **AuctionConsole** in the "Name" field
4. Enter **org.mortbay.jetty.Server** in "Superclass" field
5. Click "Finish"

Add Logic to setup embedded server

Paste the following code into the AuctionConsole class:

```
private WebApplicationContext mainContext;

    public AuctionConsole(int portNum) throws MalformedURLException,
IOException, URISyntaxException {
        mainContext = new WebApplicationContext();
        mainContext.setContextPath("/");

        ContextHandlerCollection contexts = new
```

```

ContextHandlerCollection();
    this.setHandler(contexts);
    mainContext.setBaseResource(new FileResource(new
File(".").toURI().toURL()));
    contexts.addHandler(mainContext);

    Connector connector = new SelectChannelConnector();
    connector.setPort(portNum);
    addConnector(connector);
    mainContext.setConnectorNames(new String[] { connector.getName()
});
}

void addServlet(String path, Servlet handler) {
    ServletHolder requestServlet = new ServletHolder(handler);
    mainContext.addServlet(requestServlet, path);
}

```

Create custom servlet

1. Create **com.x.devcamp.bidder.ConsoleServlet** class with **javax.servlet.http.HttpServlet** as the "Superclass"
2. Add **doPost** method to handle requests:

```

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
}

```

3. In the **doPost** method, add the follow code to authenticate the request:

```

String authToken = req.getHeader(HttpHeaders.AUTHORIZATION);
if (!"Bearer
Q1VTRk9SQVVUSE1ELTEA7/GRzPeAKiymgVsONHEikg==" .equals(authToken)) {
    resp.sendError(HttpStatus.SC_UNAUTHORIZED, "Invalid Authorization
token");
    return;
}
String tenantId = req.getHeader("X-XC-TENANT-ID");

```

4. Add helper method to **ConsoleServlet** convert HTTP request data into a **byte[]**:

```

private static byte[] getMessageBody(HttpServletRequest request) throws
IOException {
    int length = request.getContentLength();
    if (length < 0) {
        length = 4096;
    }
    ByteArrayOutputStream baos = new ByteArrayOutputStream(length);
    byte[] buffer = new byte[length];
    int n;
    InputStream in = request.getInputStream();
    while ((n = in.read(buffer)) > 0) {
        baos.write(buffer, 0, n);
    }
    return baos.toByteArray();
}

```

5. In the **doPost** method, after the authorization logic, add code to deserialize the incoming messages:

```
byte[] data = getMessageBody(req);
```

6. Add code to track the message ID:

```
String messageGuid = req.getHeader("X-XC-MESSAGE-GUID");
```

7. Add code to process the message based on the topic:

```
String topic = req.getRequestURI();
if (topic.indexOf("/bid/") > -1) {
    Bid bid = AvroEncDecoder.decode(data, Bid.SCHEMA$);
    System.out.println "[" + messageGuid + "]" + tenantId + " " + topic + ": "
+ bid.listingId + " -> " + bid.bidAmount);
} else if (topic.indexOf("/auction/") > -1) {
    Listing listing = AvroEncDecoder.decode(data, Listing.SCHEMA$);
    System.out.println "[" + messageGuid + "]" + tenantId + " " + topic + ": "
+ listing.xId + ":" + listing.title + " -- $" + listing.price.amount);
} else {
    System.out.println "[" + messageGuid + "]" + tenantId + " " + topic + ": "
+ "Rogue message");
}
```

8. Add a main method to AuctionConsole.java to pull it all together:

```
public static void main(String[] args) throws Exception {
    AuctionConsole server = new AuctionConsole(9000);
    server.addServlet("/*", new ConsoleServlet());
    server.start();
}
```

Run the AuctionConsole class

9. Right-click on the "AuctionConsole" class, Run As > Java application

10. You will start seeing messages like the following:

```
[a4b908f7-4338-4d87-b642-963905bcb271][null]
/auction/updated: 106:Size 3 Pink Soccer Ball -- $0.0
```

These will get more interesting after the next lab.

Lab 6: Start Publishing Messages

Create a Bidder class

1. Using some of the steps from the previous lab, create a **com.x.devcamp.bidder.Bidder** class

2. Add a main method to the Bidder class:

```
public static void main(String[] args) throws Exception {
}
```

3. Add the following to the top of the main method to collect information for the bid:

```
if (args.length != 3) {
    System.err.println("usage: java com.x.devcamp.bidder.Bidder tenant
listingId bidAmount");
    return;
}
String tenant = args[0];
String listingId = args[1];
BigDecimal amount = new BigDecimal(args[2]);
```

4. Create a Bid based on the provided data:

```

Bid bid = new Bid();
bid.bidAmount = new CurrencyAmount();
bid.bidAmount.amount = amount.doubleValue();
bid.bidAmount.code = "USD";
bid.listingId = listingId;

```

5. Add the following line, which encodes the Bid using Avro:

```

byte[] data = AvroEncDecoder.encode(bid, Bid.SCHEMA$);

```

6. Convert tenant to tenant token:

```

String tenantAuth = null;
if ("bidder1".equals(tenant)) {
    tenantAuth = "QVVUSE1ELTEAuRyP8LTHzE/oooUzVdZZdQ==";
} else {
    tenantAuth = "QVVUSE1ELTEAS9f0EZZjuvH+w7wQP/KM1A==";
}

```

7. Publish Bid message

```

String messageGuid = new XFabricPublisher("").publishMessage("/bid/placed",
tenantAuth, data);
System.out.println "[" + messageGuid + "] Placed $" + amount + " bid for "
+ listingId + " on behalf of " + tenantAuth);

```

Run Bidder placing an initial bid

1. Right-click the "Bidder" class > Run As > Java Application
2. The console will output the usage information
3. Right-click the "Bidder" class again > Run As > Run Configurations..
4. In the "Name" field, enter **Bidder 1**
5. Click the "Arguments" tab
6. In the Program Arguments field, enter: **bidder1 114 5**
Note: Replace '114' with whatever listing ID is currently on the block in your AuctionConsole window.
7. In the AuctionConsole window, you will see a message indicating that the bid was accepted.

Run Bidder again with another bid

1. Right-click on the "Bidder" class > Run As > Run Configurations..
2. Right-click on the "Bidder 1" configuration, and select the Duplicate option
3. Change the "Name" field to **Bidder 2**
4. Click on the "Arguments" tab
5. In the Program Arguments field, enter: **bidder2 114 10**
6. The AuctionConsole window will now show the new bid being accepted and the original bid being outbid.